DeepRob Final Report
iSDF: Real-Time Neural Signed Distance Fields for Robot Perception Fetch Extension

Billy Mazotti
Robotics Engineering
University of Michigan
Ann Arbor, United States
bmazotti@umich.edu

Broderick Riopelle
Electrical and Computer Engineering
University of Michigan
Ann Arbor, United States
broderio@umich.edu

Gregor Limstrom
Robotics Engineering
University of Michigan
Ann Arbor, United States
limstrom@umich.edu

***Abstract*—This paper details our final project report for ROB599: DeepROB at Univeristy of Michigan. We chose to extend the iSDF paper by running the algorithm on data collected by the PROGRESS Lab's Fetch robot at the Ford Robotics Building. In this paper, we will introduce our problem, cover the related work, detail our experimental setup, and give our results and conclusions. You can find our project code at: https://github.com/BillyMazotti/iSDF_ROB599.**

I. Introduction

A common problem in robotics is the efficient generation of a map of the surroundings for robots to operate within, including tasks such as motion planning, obstacle avoidance, and path generation.

Robotic maps are typically stored in Signed Distance Fields (SDFs), scalar fields that associate each point in 3D space with the signed distance to the closest surface point. SDFs provide an efficient representation of the environmental model for collision avoidance and trajectory optimization. However, SDFs are usually considered too expensive to compute in real-time and are pre-calculated, or a shortened or partial SDF is used to meet the computational requirements of the real-time robotic system.

Euclidean Signed Distance Fields (ESDFs) compute an unoccupied point's distance as the Euclidean distance to the nearest occupied point in a pre-determined map. These SDFs are calculated using expensive wavefront propagation algorithms and scale poorly with large spaces because each point must be calculated based on other point's distance in the field.

Truncated Signed Distance Fields (TSDFs) compute a point's distance as the distance to the surface along the ray direction from the center of a sensor sample. TSDFs are used in SLAM applications, are computationally efficient, but must be cut off farther from the surface of objects as the sampling can lead to large errors further from surfaces where sensor samples may be sparse. These are efficient and can run in real-time on most robots.

ESDFs and TSDFs are typically stored in Voxel grids, which represent a discretized volumetric approximation of a scene. However, these grids tend to be limited in resolution due to poor time and space complexity and suffer from an inability to represent smaller color and geometry variations to meet real-time constraints in larger datasets. These grids are also very memory-intensive as each point must be stored individually.

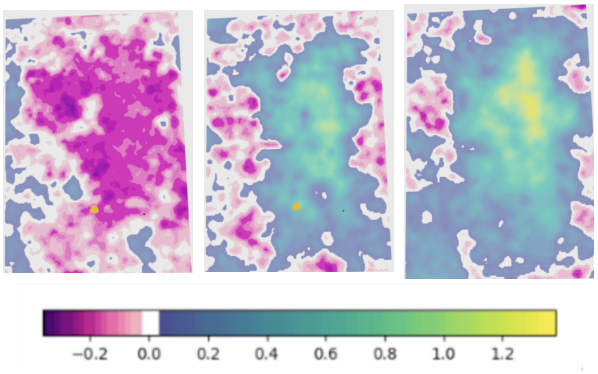

Fig. 1. From left to right, SDF slices with 500 steps training of the PROGRESS LAB at heights around 0%, 50% and 100% of the recorded environment height

In recent years, neural radiance fields have emerged as an alternative, in which the volumetric space representation is stored in a learned distance function. These are quick to train, more memory-efficient, and use positional embedding to represent small color and geometry variations [1]. The function is created using an inexpensive Multi-Layer Perceptron that serves as an N-degree equation for the state space representation.

Our final report explores iSDF, a recent paper that extends NeRFs to efficiently and quickly generate a mesh model of the environment in real-time with varying resolution, in a fraction of the memory footprint of other modern methods [5].

We extend iSDF by integrating it with the PROGRESS Lab's Fetch robot. This involves integrating ORBSLAM3 into the Fetch ROS stack, modifying the iSDF nodes to accept the correct data format from Fetch's camera and depth sensors, and collecting live data from inside the Ford Robotics Building.

II. Related Work

Full SDF reconstruction before iSDF was assumed to be too computationally expensive for real time applications, and as a result was not thoroughly explored.

Current state of the art approaches include Voxblox, a method which takes in RGB-D with pose, which is fed into a TSDF layer, which is then corrected by an ESDF layer before being stored in a voxel hash, which is a dynamically resized voxel grid capable of capturing smaller details and achieving more computationally efficient performance [2]. FIESTA is a second method which also uses RGB-D with pose, which is used to build an Occupancy Map of the space, which is then filled in and corrected by an ESDF layer [3]. Both of these methods use ESDFs to fill in unsampled data, which as mentioned before is computationally prohibitive, and limits the resolution of the environmental representation.

iMAP, or Implicit Mapping and Positioning in Real-Time [4] is another key enabling algorithm for iSDF. iMAP was the first paper to achieve a scene representation using a neural field. It accomplishes this through adaptive keyframe selection by calculating the information gain of each frame and only training on the sparse, but valuable data. This significantly cuts down on the number of frames in the pipeline and allows real time training and reconstruction. In addition, iMAP also uses active image point sampling to solve the problem of network forgetting, and preserve the older reconstruction as new data and training frames are brought in to the machine learning pipeline.

The iSDF network brings all these advancements together. It uses NeRFs with the active keyframe selection from iMAP in order to efficiently predict the full SDF of an environment by training an MLP with 4 hidden layers each with 256 activations and softplus between each intermediate layer [5]. The resulting equation allows full 3D reconstruction of rooms with predictive filling for under-observed locations, resulting in a full mesh, whereas Voxblox and Fiesta leave low sampled areas empty. iSDF was able to achieve comparable results with the state of the art when close to a surface, and significantly better results further from a surface thanks to the predictive nature, all while having a 5-30x smaller memory footprint and realtime performance in full SDF reconstruction.

III. Algorithmic Extension

Prior to working with iSDF's source code, our team proposed three different ideas for extending iSDF:

1) Run iSDF on new data collected by a Fetch Robot to observe edge-cases and real-world implications
2) Incorporate semantic segmentation to provide additional scene information for higher-level trajectory optimization
3) Integrate iSDF into a planner to solve engineering challenges with downstream implementation

After encountering several installation and running issues with the source code, we decided to focus on making idea number 1 the focus of our algorithmic extension. In this section, we first discuss discuss the efforts that went into setting-up the virtual machine used in reproducing results from the paper, then provide an overview of our algorithmic extension.

Fig. 2. Fetch Robot provided by the PROGRESS Lab at the University of Michigan, Ann Arbor.

A. Virtual Machine Set-up

The authors of iSDF ran all experiments using Ubuntu 20.04 on a machine with an Intel i7-8700K CPU and a GeForce RTX 2080 GPU. Our project was ran on a MacBook Pro with 32GB of unified memory and the Apple M1 Max chip. To run Ubuntu 20.04, we used VMware Fusion and found best performance when allocating 8 of our 10 available CPU cores and 12GB of RAM.

Following the source code for setting up iSDF to run on ReplicaCAD's simulated environment sequences, we encountered errors concerning conflicting package versions while trying to build the necessary conda environment. To solve this issue we created a new environment.yml file for automatically creating the necessary conda environment. The environment.yml file adds conda-forge as a build channel, removes all build channel dependencies used by the authors of iSDF, and similarly removes version requirements for all packages except *pip*, *python*, *pyglet*, and *setuptools*.

We have included the new environment.yml file for our specific setup as well as the environment file describe above. While PyTorch has released support for GPU speed-up with the Apple M1 on Mac OS, we ran everything off CPU due to current lack of known solutions for PyTorch support of the Apple M1 on Ubuntu OS. As a result, the combination of VMware Fusion's student availability, the new environment.yml file, and removing the need fora GPU allows makes working on this project available to more students at the cost of SDF and training time performance as discussed in Section IV.

B. Collecting Real-World Test Data

For our algorithmic extension, we chose to run iSDF on a live data set collected from the University of Michigan's PROGRESS Lab's Fetch robot. Datasets capable of running on iSDF require synchronized RGB images, depth images, and pose. To collect this data, we recorded a rosbag of Fetch's IMU, depth, and RGB camera data. In order to integrate iSDF

into the Fetch dataset, we had to make several modifications to the ROS nodes used in the algorithm.

First, iSDF uses ORBSLAM3 to generate a ”frame” message, which is a ROS message with an RGB image, a depth image, and a pose embedded within. The source code for iSDF uses a ros-wrapper around ORB-SLAM3 to take in rostopics for monocular RGB images, depth images, and IMU data and output the frame topic required by iSDF. Based on the source code's existing architecture for running iSDF on a realsense camera, we created new launch and configuration files within both the iSDF and ORB-SLAM-wrapper directories. These new files take into account the Fetch's RGB camera sensor intrinsic values obtained from the robot's topic ”/camera_information”.

All config and launch files that were customized to running on fetch can be found in a folder in the main directory of our project code. To streamline the setup process for testing on fetch and to serve as a guide for other researchers trying to run iSDF on their own rosbag files or robots, we have created a bash script that to create copies of all customized config and launch files in their necessary directories.

Lastly, due to hardware limitations we slowed down the rosbag to 0.1 times real-time to ensure that enough keyframes were captured during the experimentation process.

IV. Experiments and Results

In this section, we present the results of our experiments that extended the iSDF network to real-world environments using the Fetch robot. The experiments were conducted on two different environments, FRB 2020 and FRB 2150, both of which were complex and included multiple obstacles such as tables, chairs, and people. During the data collection, different Fetch robots were employed, and while they were of the same model, the robot used in the first trial experienced a wheel issue that resulted in wobbling and vibration during data collection. Therefore, we neglected this dataset in our experiments.

Once the raw sensor data was collected from the Fetch robot's RGBD camera and IMU, we utilized the ORB-SLAM3 algorithm to generate pose messages, which were published to the iSDF ROS node for training the network. This allowed us to generate a more accurate representation of the robot's motion and pose compared to using the raw odometry information from Fetch.

The authors trained their network for 65 seconds at approximately 33ms per step, so we allowed the network to train for 2000 steps to replicate a similar level of training that was completed in the paper.

A. Experimental Setup

Three experiments were conducted to evaluate the effectiveness of our approach. The first two experiments tested the *apt_2_nav* environment provided by ReplicaCAD used by the authors of iSDF, with the intention of replicating the results of the paper. The first experiment was an incremental training of the network, which involved training the iSDF network on the raw data and allowing it to learn the keyframes from start. The second experiment was a non-incremental training of the network, which involved running the iSDF network with the keyframes generated by the authors' model preloaded. These keyframes are shown in Fig. **??**. This led to a faster training of the network. Due to hardware limitations and lack of access to a GPU, this method was chosen to replicate the results of the paper.

The third experiment involved an incremental training of the network on the dataset collected from the second trial on the Fetch robot. The rosbag of the collected Fetch data was played back through ORB-SLAM3 at 0.1x speed to increase the number of captured keyframes and decrease the likelihood of ORB-SLAM3 failing due to sharp rotations in the trajectory.

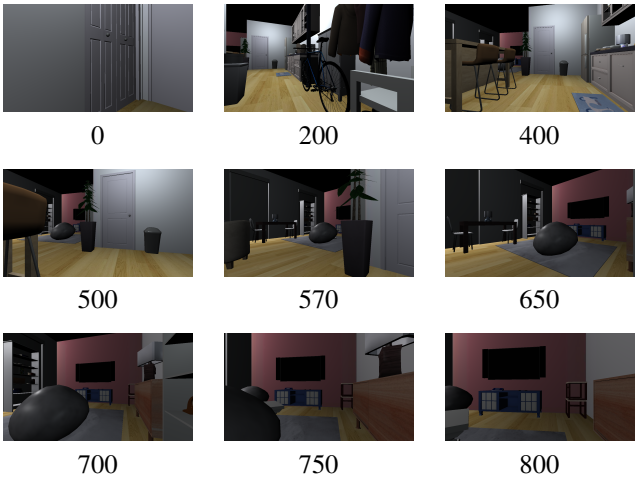

Fig. 3. RGB images of the keyframes used during non-incremental SDF generation for the *apt_2_nav* trajectory

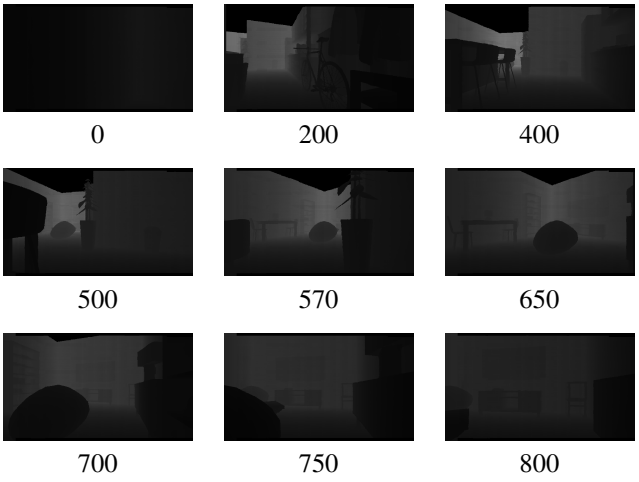

Fig. 4. Depth iamges of the keyframes used during non-incremental SDF generation for the *apt_2_nav* trajectory

B. Results

In the first experiment, we trained the iSDF network incrementally on the *apt_2_nav* environment dataset. The generated

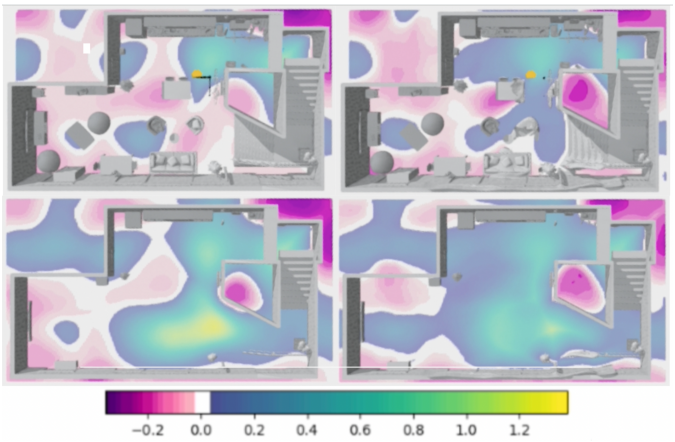

Fig. 5. SDF slices from incremental test on *apt_2_nav* clockwise from top left: floor height at 50 steps, floor height at 500 steps, half room height at 500 steps, half room height at 50 steps

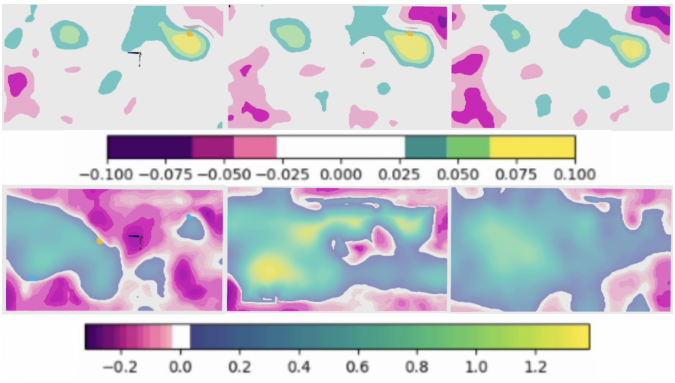

Fig. 6. SDF slices during incremental training: from left to right, slices are taken at 0%, 50% and 100% height, and from top to bottom, the slices represent the SDF at 5 and 500 steps into training the MLP.

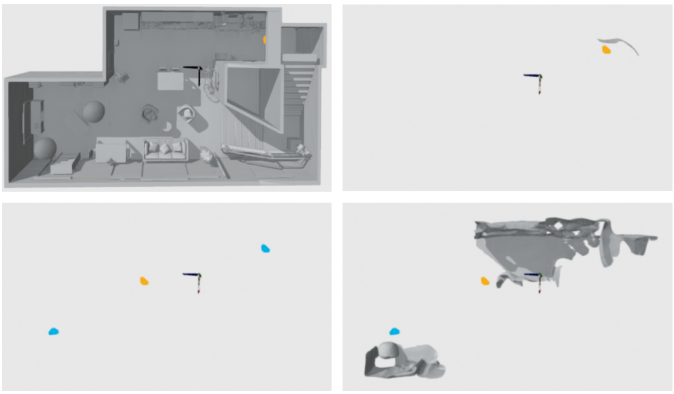

Fig. 7. Clockwise from top left: ground truth environment, keyframe locations and generated mesh at 5 steps, generated mesh at 500 steps, and incremental keyframes used.

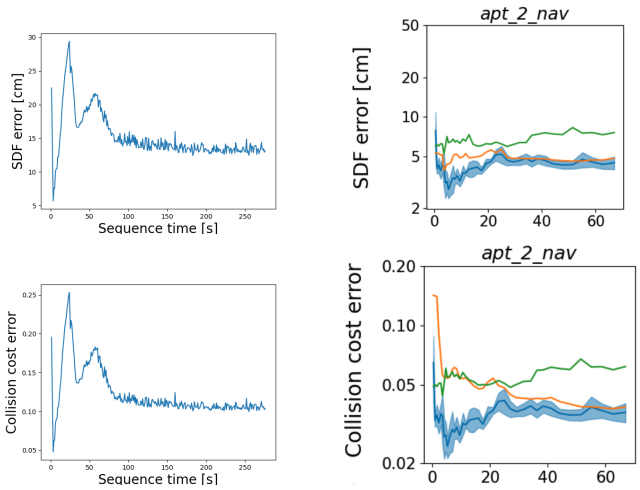

Fig. 8. Comparison of non-incremental reproduction (left) and paper reported (right) SDF error and collision cost error

SDF models after 50 and 500 steps of training are compared in Figure 5. As training progresses, the SDF generated by the network appears to fit more accurately to the ground truth mesh of the scene. The evolution of the SDF from step 5 to step 500 is depicted in Figure 6. We note that the network initializes the unexplored environment as a surface. As more of the scene is explored and keyframes are added, the network learns which space is actually occupied. However, due to training incrementally, only 3 out of the 9 keyframes reported in the original paper were found and used for training, as illustrated in Figure 7.

The second experiment was conducted to address the minimal number of generated keyframes from the first experiment's incremental training. This involved training the network non-incrementally with all nine keyframes preloaded into the network. The progression of the generated mesh as the network trains is illustrated by Figure 9. We notice that as the network trains, the mesh aligns more closely with the ground truth environment observed by the camera view. This is corroborated by the data in Figure 13, which increases in accuracy as the number of training steps increase. The reproduced network was evaluated on its SDF error and collision cost error, which were found to be 14cm and 0.12, respectively, as shown in Figure 8. In comparison, the authors of iSDF reported an SDF error of under 5cm and a collision cost error of about 0.04. We note that the published software was used without modification during training, and the discrepancy in results can be explained by computational limitations. While the authors had access to several GPUs, we were limited to CPU use. Nevertheless, a similar trend between SDF error and collision cost error was observed in both the reproduced and reported results. Although the reproduced results were worse, they were within an order of magnitude of the reported results.

The third experiment aimed to evaluate the feasibility of extending iSDF into real-world environments. The path taken by the Fetch robot within the PROGRESS Lab, as output from the ORB-SLAM3 algorithm, is presented in Figures 10a and 10b. Figure 10c shows the location of the keyframes generated by iSDF, while Figure 10d displays the corresponding keyframe images. Throughout the experiment, a total of 10 keyframes were generated. Figure 1 illustrates the generated SDF at